Communication Response Time in P-NET Networks: Worst-Case Analysis Considering the Actual Token Utilisation[1]

Eduardo Tovar‡, Francisco Vasques†, Alan Burnsξ

‡ Department of Computer Engineering, Polytechnic Institute of Porto, Rua de São Tomé, 4200 Porto, Portugal
emt@dei.isep.ipp.pt
† Department of Mechanical Engineering, University of Porto, Rua dos Bragas, 4099 Porto Codex, Portugal
vasques@fe.up.pt
ξ Department of Computer Science, University of York, Heslington, York, YO1 5DD, England
buns@cs.york.ac.uk

(January 1999)

Abstract. Fieldbus networks aim at the interconnection of field devices such as sensors, actuators and small controllers. Therefore, they are an effective technology upon which Distributed Computer Controlled Systems (DCCS) can be built. DCCS impose strict timeliness requirements to the communication network. In essence, by timeliness requirements we mean that traffic must be sent and received within a bounded interval, otherwise a timing fault is said to occur. P-NET is a multi-master fieldbus standard based on a virtual token passing scheme. In P-NET each master is allowed to transmit only one message per token visit, which means that in the worst-case the communication response time could be derived considering that the token is fully utilised by all stations. However, such analysis can be proved to be quite pessimistic. In this paper we propose a more sophisticated P-NET timing analysis model, which considers the actual token utilisation by different masters. The major contribution of this model is to provide a less pessimistic, and thus more accurate, analysis for the evaluation of the worst-case communication response time in P-NET fieldbus networks.

1 Introduction

In the past decade manufacturing schemes have changed dramatically. In particular, the CIM (Computer Integrated Manufacturing) concept has been stressed as a means of achieving greater production competitiveness. The driving forces behind the changes also resulted from the increased development and utilisation of new technologies that make massive use of microprocessor-based equipment.

[1] This work was partially done during 3 visits of Eduardo Tovar to the Real-Time Systems Research Group of the Computer Science Department of the University of York, and partially funded by ISEP, under the project REMETER, by FLAD, under the Project SISTER 471/97 and by FEUP, under the research programme PRODEP.

Integration implies that the different subsystems of the manufacturing environment interact and co-operate with each other. This means transfer, storage and processing of information in a widespread environment. In other words efficient support for data communications is required.

Nowadays, communication networks are available to virtually every aspect of the manufacturing environment, ranging from the production planning to the field level. However, the use of communication networks at the field level is a much more recent trend. Indeed, only more recently network interfaces become cost-effective for the interconnection of devices such as sensors and actuators, which, in the majority of the cases, are expected to be cheaper than the equipment (e.g., workstations and numerically-controlled machines) typically interconnected at upper control levels of the manufacturing environment.

The field level includes the process-relevant field devices, such as sensors and actuators. The process control level is hierarchically located above the field level, and directly influences it in the form of control signals. If the control is computer-based, the process control level uses the data received from the sensors to compute new commands, which are then transmitted to the actuators.

Most of the computer-controlled systems are also real-time systems. Real-time computer systems are defined as those systems in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced (Stankovic, 1988). For instance, assume that one of the inputs of the computer system is a Boolean information of an alarm condition. The computer system must be able to handle such alarm condition (process that input and produce outputs accordingly), within a bounded time span. Thus, a computer system not only must react to stimuli of the controlled object, which in essence means the provision of new commands based on the current state of the controlled object, but emphatically it must react to stimuli of the controlled object within time intervals dictated by its environment.

A computer-controlled system may have a centralised architecture. By centralised architecture we mean that there is only one single computer system unit, which has I/O capabilities for supporting both the instrumentation interface (to interface the computer system with the controlled object) and the man-machine interface (to interface the computer system with the operator). In the case of a centralised architecture, the sensors and actuators are connected to the computer system via point-to-point links. Figure 1a) illustrates such kind of architecture.

There are several advantages in using a field level communication network as a replacement for the point-to-point links between sensors/actuators and the computer system. The main advantage is an economical one. Indeed, this is perhaps its single best advantage. As depicted by figure 1, a cost reduction can be obtained by replacing a significant part of the wiring by a single wire, and by using simpler network architectures and protocols than the used in networks for the upper levels of the control hierarchy. Of course, the use of a single wire brings also easier installation and maintenance, easier detection and localisation of cable faults, and it provides an easier expansion due to the modular nature of the network.

Typically, a field level network will be a broadcast network, where several network nodes share a common communication channel. Messages are transmitted from a source network node to a destination network node via the shared communication channel. In fact, broadcast networks are commonly used in most types of local area

networks (LANs). A major problem occurs when at least two network nodes attempt to send messages via the shared channel at about the same time. This problem is solved by a medium access control (MAC) protocol. Independently of how this protocol is implemented, medium access contention occurs, as a message must contend both with other messages from the same network node as well with messages from other network nodes. Therefore, to cope with the real-time requirements of the controlled applications, the field level network must guarantee a bounded access time, or, in other words, the MAC protocol must be deterministic.

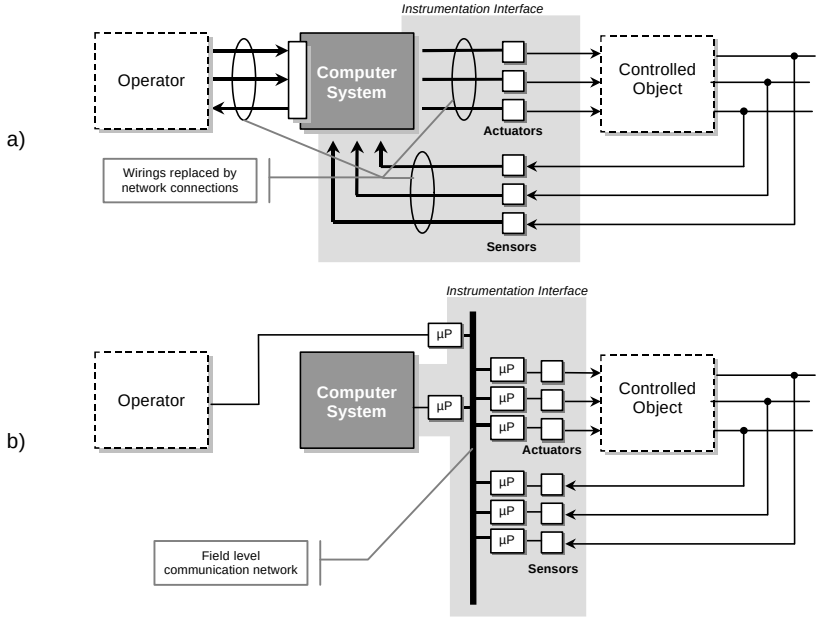

Fig. 1. Centralised (a) and Decentralised (b) Computer-Controlled Architectures

During the past decade or so, a significant number of field level networks, usually called fieldbus networks, have been proposed to support distributed[2] computer--controlled systems (DCCS). Some distinguished examples are FIP (Fip, 1990), PROFIBUS (Profibus, 1992), CAN (Sae, 1992) and P-NET (P-net, 1996). In parallel, several international standardisation efforts have been, and are still being carried out. One of the most relevant resulted into the European Standard EN 50170 (Cenelec, 1996), which basically encompasses three different fieldbus profiles: FIP, PROFIBUS and P-NET.

In FIP, the determinism is guaranteed by a bus arbitrator, which, for periodic traffic, controls data transfers according a static scanning table. The real-time capabilities of FIP have been extensively studied (Pedro and Burns, 1997, Raja *et al.*, 1995). PROFIBUS adopts a simplified version of the timed token (TT) protocol

[2] In both centralised and decentralised computer-controlled architectures all the control algorithms are implemented in the single computer system. Contrarily, in a distributed computer-controlled architecture the tasks of the control algorithms are actually distributed by several computing nodes.

(Grow, 1982). Despite some differences to the TT protocol used in FDDI or IEEE802.4, for which real-time characterisation have been deeply addressed - (Agrawal *et al.*, 1992; Montuschi *et al.*, 1992) are just some examples -, it is still possible to guarantee real-time behaviour with PROFIBUS networks (Tovar and Vasques, 1998a, Tovar and Vasques, 1998b). CAN is itself a priority bus, which adopts a collision avoidance version of the well-known CSMA (Carrier Sense with Multiple Access) MAC protocol. In (Tindell *et al.*, 1994; Tindell *et al.*, 1995) the authors showed how it is possible to guarantee real-time behaviour with CAN networks.

P-NET also offers a deterministic access. P-NET adopts a virtual token passing (VTP) scheme. Although this is not much relevant for the timing behaviour of this MAC approach, it worth mentioning that, conversely to other token passing schemes, in P-NET there is no explicit token transmission between stations. The determinism is not achieved by means of controlling the token rotation time, as it happens in networks based on the TT protocol. Instead, the bounded access delay is implicitly guaranteed by the fact that at each token visit only one message request may be performed.

In (Tovar *et al.*, 1998c) the authors analyse the P-NET's MAC behaviour and propose a worst-case response time analysis for P-NET messages. We now improve such analysis by considering the actual token utilisation, instead of considering always the worst-case token rotation time. The major contribution of this paper is that we provide a much less pessimistic and more accurate analysis for the evaluation of the worst-case response time in P-NET fieldbus networks.

The remaining of this paper is organised as follows. In section 2 we briefly describe the P-NET's MAC protocol. In section 3 we provide a basic worst-case response time analysis, where the actual token utilisation is not considered. In section 4, we describe how the actual token utilisation can be taken into account for the evaluation of the worst-case response time, by proper use of the information about the message request periodicities in all masters. Finally, in section 5 we give conclusions.

2. Basic Concepts on P-NET's MAC Protocol

The name P-NET is a derivation of "Process Network". P-NET was designed as a communications link between distributed process control sensors, actuators and small programmable controllers, and has recently gained an increased role, as it became, along with PROFIBUS and FIP, a European Standard, the EN 50170 - General Purpose Field Bus Communication System.

P-NET is a multi-master standard. Therefore, all communication is based on a principle, where a master sends a request and the addressed slave immediately returns a response. For multi-master support, P-NET uses a Virtual Token Passing (VTP) scheme. Figure 2 illustrates the hybrid-operating mode of the P-NET's MAC.

The VTP scheme is implemented using two protocol counters. The first one, the Access Counter (AC), holds the node address of the currently transmitting master. When a request has been completed and the bus has been idle for 40 bit periods

(520μs @ 76,8Kbps[3]), each one of the ACs is incremented by one. The master whose AC value equals its own unique node address is said to hold the token, and is allowed to access the bus. When the AC is incremented as it exceeds the "maximum No of Masters", the AC in each master is pre-set to one. This allows the first master in the cycling chain to gain access again.

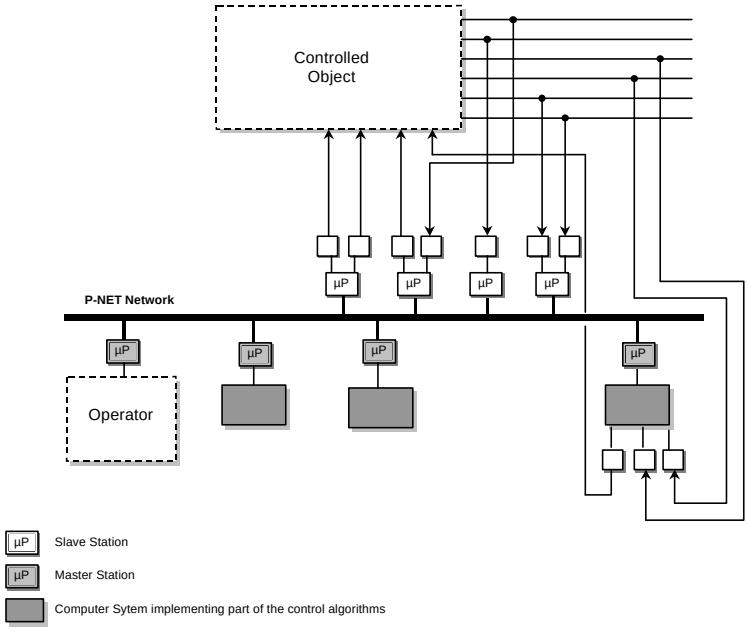

Fig. 2. Example of Multi-Master/Slaves Organisation in a P-NET Network

The second counter, the Idle Bus Bit Period Counter (IBBPC), increments for each inactive bus bit period. Should any transactions occur, the counter is re-set to zero. As explained above, when the bus has been idle for 40 bit periods following a transfer, all ACs are incremented by one, and the next master is thus allowed to access the bus.

If a master have nothing to transmit (or indeed is not even present), the bus will continue inactive. Following a further period of 130μs (10 bit periods), the IBBPC will have reached 50, (60, 70,…) all the ACs will again be incremented, allowing the next master access. The virtual token passing will continue every 130μs, until a master does require access.

The P-NET standard also stands that each master is only allowed to perform one message transaction (later on defined as message cycle) per token visit. This is an important notion for the remaining of the paper.

[3] The P-NET standard uses a data rate of 76800 bps. This data rate resulted from weighing up the conflicting requirement for data to be transported as fast as possible, but not at such speed as to negate the use of standard microprocessor UARTS, or restrict the usable distance or cable type (Jenkins, 1997).

After receiving the token, the master must transmit a request before a certain time has elapsed. This is denoted as the master's reaction time, and the standard imposes a worst-case value up to 7 bit periods).

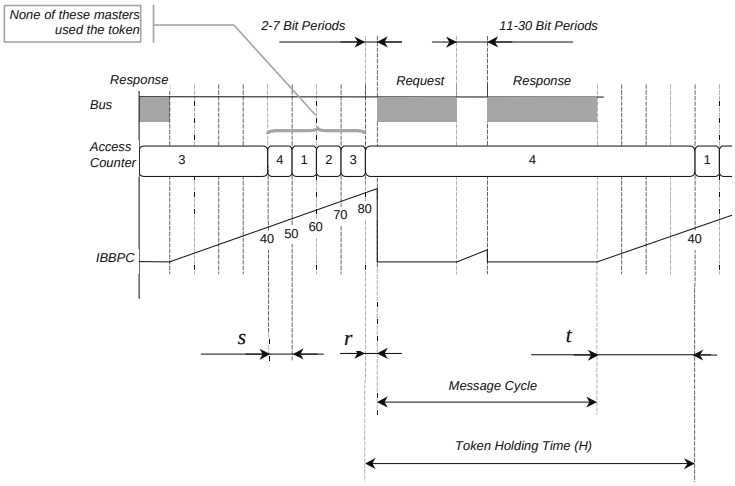

Fig. 3. P-NET's Virtual Token Passing Protocol

A slave is allowed to access the bus, between 11 and 30 bit periods after receiving a request, measured from the beginning of the stop bit in the last byte of the frame. The maximum allowed delay is then 390µs (corresponding to 30 bit periods). Later on, this delay will also be denoted as the slave's turnaround time.

As already stressed, at each token visit, a master may perform at most one message cycle. A message cycle is composed by a master's request followed by the addressed slave's response.

Assume that C_M is the maximum transmission duration of all message cycles in a P-NET network. This duration includes both the longest request and response transmission times, and also the worst-case slave's turnaround time.

If a master uses the token to perform a message cycle, we can define a token holding time[4] as:

$$H = r + C_M + t \qquad \textbf{(1)}$$

In equation (1), t (= 40 bit periods) corresponds to the time to pass the token after a message cycle has been performed. r (≤ 7 bit periods) denotes the worst-case master's reaction time. If a station does not use the token to perform a message cycle, the bus will be idle during s (= 10 bit periods) before all ACs are incremented. For better understanding both the basic MAC procedures and the notation used, refer to figure 3.

At a glance, the message flow in a P-NET network can be compared to a switch with n input queues (each one corresponding to one P-NET master), and with only

[4] It is not usual to include the token passing time in the token holding time. However, due to the specificity of the Virtual Token Passing scheme, we decided to associate the token holding time with the state of the P-NET access counter in each node.

one output. Each one of the input queues is served in a purely round-robin fashion, one message at a time. Within each queue, messages are served in a first-come-first-served (FCFS) fashion. Figure 4 depicts this analogy.

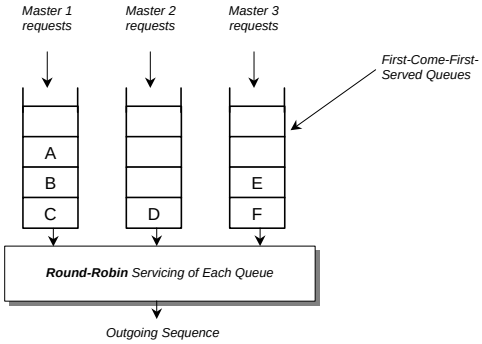

Fig. 4. Queuing Analogy of the P-NET's VTP Protocol

Consider the example contained in figure 4. Assume that the token is passed to master 1, and no other requests will be made in masters 2 and 3 within a relatively long time. In this case the output sequence would be as follows:

H_C, H_D, H_F, H_B, σ, H_E, H_A, σ, σ, …

where H_C ($H_{A, ...}$) corresponds to the token holding time resulting from transmitting the message request C (A, …) and receive the associated response. Figure 5 is a Gantt chart of the outgoing sequence resulting from the above example.

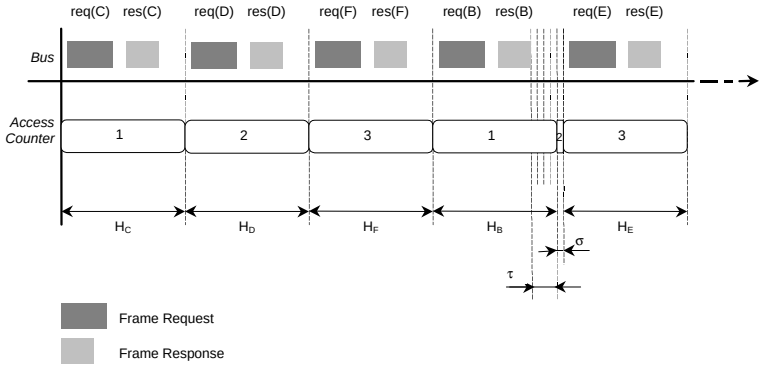

Fig. 5. Gantt Chart of the Example

Note that *s* (= 10 *bp*) is normally much shorter than *H*, since *H* includes both the request and response frames. In fact, a P-NET frame[5] is as illustrated in figure 6. As in P-NET each frame byte actually corresponds to 11 bits[6], a frame may have up to

[5] The Node Address Field may have up to 24 frame bytes. P-NET uses these complex addresses if multiple segments are used and special devices are used to relay frames between the different segments.

[6] In P-NET all the frame bytes are sent asynchronously, with one start bit (logical zero), 8 data bits (with LSB first), one address/data bit and one stop bit. Within a frame, a start bit must immediately follow a stop bit.

759 bits (69 x 11 bits). Thus, considering the case that both the request and response frames have 759 bits (more realistically either the request will be longer - case of writing data to a slave - or the response will be longer - case of receiving data from a slave), the overall sum for the token holding (*H*) time may go up to 1595 bit periods, corresponding to 20.8 ms @ 76800 bps.

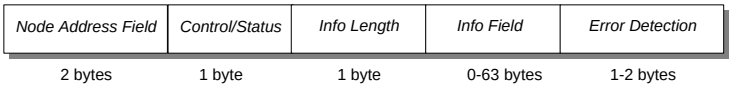

Fig. 6. Typical P-NET Frame

Table 1 gives the worst-case duration of the token holding time (*H*) in P-NET, with the explicit weight of the different contributing components.

Table 1. Worst-Case Duration for *H*

Component	**Worst-Case Duration** (in bit periods)
Master's Reaction Time (r)	7
Request Transmission Time	759
Slave's Turnaround Time	30
Response Transmission Time	759
Token Passing (t)	40
Total	**1595**

3. Basic Timing Analysis

After queuing a request in a master's outgoing queue, will the message cycle be performed before its deadline? This is the main question that must be addressed by the pre-run-time schedulability analysis of P-NET networks.

In this section we present a basic pre-run-time schedulability analysis, which is based on the worst-case token utilisation, that is, which assumes a fully utilisation of the token.

3.1. Network and Message Models

We consider a network with *n* masters, with addresses ranging from 1 to *n*. Each master accesses the network according to the VTP scheme. Hence, first master 1, then master 2, 3, … until master 1, and then again 2, 3, … Slaves will have network addresses higher than *n*.

We also assume the following message stream model:

$$S_i^k = (C_i^k, T_i^k, D_i^k) \quad \textbf{(2)}$$

S_i^k defines a message stream *i* in master *k* (*k* = 1, .., *n*). A message stream is a temporal sequence of message cycles concerning, for instance, the remote reading of

a specific process variable. C_i^k is the longest message cycle duration of stream S_i^k. T_i^k is the periodicity[7] of stream S_i^k requests. Finally, D_i^k is the relative deadline of the message cycle, that is, the maximum admissible time span between the instant when the message request is placed in the outgoing queue and the complete reception of the related response at the master's incoming queue. We consider that messages generated in the distributed system can be periodic or sporadic. For the case of sporadic message requests, its period corresponds to the minimum time between any two consecutive requests for that stream. ns^k is the number of message streams associated with a master k.

In our model the relative deadline of a message can be equal or shorter than its period ($D_i^k \leq T_i^k$). Thus, if in the outgoing queue there are two message requests from the same message stream, this means that a deadline for the first of the requests was missed[8]. It also results that the maximum number of pending requests in the outgoing queue will be, in the worst-case, ns^k.

We denote the worst-case response time of a message stream i in a master k as R_i^k. This time is measured starting at the instant when the request is placed in the outgoing queue, until the instant when the response is completely received at the incoming queue. Basically, this time span is made up of the two following components:

1. the time spent by the request in the outgoing queue, until gaining access to the bus (queuing delay);
2. the time needed to process the message cycle, that is, to send the request and receive the related response (transmission delay)[9].

Thus,

$$R_i^k = Q_i^k + C_i^k \quad \textbf{(3)}$$

where Q_i^k is the worst-case queuing delay of a message stream i in a master k.

In order to have simpler and more understandable analysis, we will use the maximum token holding time (see equation (1)) for all message cycle transactions, instead of considering the actual length for each particular message cycle. At the end of the paper we will update formulae in order to consider the actual length of message cycles.

Thus, we will use equation (4), instead of equation (3) to define the worst-case response time for a message request belonging to stream S_i^k:

$$R_i^k = Q_i^k + C_M \quad \textbf{(4)}$$

As we will also show, it also results from considering C_M instead of S_i^k that $Q_i^k = Q^k$, $\forall_i$ and $R_i^k = R^k$, $\forall_i$.

---

[7] In order to have a subsequent timing analysis independent from the model of the tasks at the application process level, assume that this periodicity is the minimum interval between any two requests for that stream. Otherwise, a message release jitter (Tindell *et al.*, 1995) would need to be considered.

[8] Actually, we can be more precise saying that deadlines will be missed if a new request appears, in the outgoing queue, before the completion of a previous message cycle for the same request.

[9] As the bit rate in P-NET is 76800 bps, the propagation delay can be neglected, even for P-NET networks with length of some kilometers.

3.2. Basic Analysis for the Worst-Case Response Time

A basic analysis for the worst-case response time can be performed if the worst-case token rotation time is assumed for all token cycles. As the token rotation time is the time span between two consecutive visits of the token to a particular station, the worst-case token rotation time, denoted as V, is:

$$V = n \times H \tag{5}$$

with H as defined in (1), and gives the worst-case time interval between consecutive token visits to any master k (k = 1, .., n).

In P-NET, the outgoing queue is implemented as a FCFS queue. Therefore, a message request can be in any position within the ns^k pending requests. ns^k is also the maximum number of requests which, at any time, are pending in the master k outgoing queue. This results from the adopted message stream model, which considers $D_i^k \leq T_i^k$. Hence, the maximum number of token visits to process a message request in a master k, is ns^k.

The worst-case queuing delay occurs if ns^k requests are placed in the outgoing queue just after a message cycle was completed (at the beginning of the token passing interval: t) and the token is fully utilised in the next ns^k consecutive token cycles. We denote this time instant as t_c. We consider that a message cycle was just completed since the token passing time is t (= 40 *bp*) instead of s (= 10 *bp*). Considering t leads to the largest time span till the next visit of the token to that same master k. Only then master k will be able to process the first of the ns^k requests placed in the outgoing queue at t_c.

Definition 1: *Master's Critical Instant – We define the critical instant in master k, as the instant when ns^k requests are placed in its outgoing queue just after it has completed a previous message cycle.*

Note that we can not consider releasing ns^k new requests while master k is processing a message cycle, since that would mean a deadline violation in master k (a new request released before the completion of a previous one of the same stream). If there was no message cycle being processed, there was no point in considering an earlier release time, since one of those ns^k requests would be processed in that visit.

Due to both the deadline restriction and the FCFS behaviour of the outgoing queue, none additional request can appear in master k till the time instant (t_e), when the last of the ns^k requests, made at t_c, is completely processed, otherwise, message deadline could be missed. Therefore, we introduce definition (2) and theorem (1).

Definition 2: *Master's Busy Period – We define the busy period in master k, as the time span between the critical instant, t_c, and the time instant t_e, when the last of the ns^k requests is completely processed.*

Theorem 1: *In P-NET networks, the worst-case response time of a master's message request corresponds to the longest busy period in such master.*

Proof:

The busy period starts when a critical instant occurs. By the critical instant definition, ns^k requests are placed in the outgoing queue at the earliest possible instant. As the end of the busy period is defined as being the time instant t_e, when the last of those ns^k requests is completely processed, the difference t_c - t_e gives the worst-case response time for a message request in master k, since due to the FCFS behaviour of the outgoing queue, at t_c, a message request can be in any position, from 1st to ns^k-nd.

❑

Theorem 2: *In P-NET networks, assuming that the token is fully utilised, the worst-case response time of a message request in a master k is:*

$$R^k = ns^k \times V \quad (\mathbf{6})$$

Proof:

Assuming that the token is fully utilised, the token will take $t+(n-1)\times H$ from instant t_c until the next visit to master k. At the first visit, the token arrives at $t_2=t_c+t+n+(n-1)\times H$, and only then the master will be able to process the first of the ns^k pending requests. As only one of the ns^k message requests is processed per token visit, the token will arrive at master k only at instant $t_3=t_2+(ns^k-1)\times V$ to process the last of the ns^k requests. The time elapsed since t_c is then $t_3-t_c=t+(n-1)\times H+(ns^k-1)\times V$. As the worst-case reaction time of a master is r, the last one of the ns^k message requests will start to be transmitted with a queuing delay $Q^k=t+(n-1)\times H+(ns^k-1)\times V+r$. Note that as we are assuming $C_i^k=C_M$, $\forall_{i,k}$, the worst-case queuing delay is equal for all message requests in the same master ($Q_i^k=Q^k$, $\forall_i$). As $R_i^k=Q_i^k+C_M$, the worst-case response time for a message stream i in master k is (note that R_i^k is also equal to R^k): $R^k=t+(n-1)\times H+(ns^k-1)\times V+r+C_M$, which, considering that $H=r+C_M+t$, can be re-written as follows:

$$R^k = n\times H + \left(ns^k - 1\right)\times V = V + \left(ns^k - 1\right)\times V = ns^k \times V$$

❑

Corollary: *In P-NET networks, assuming that the token is fully utilised, the worst-case queuing delay of a message request in a master k is:*

$$Q^k = t + \left(n-1\right)\times H + \left(ns^k - 1\right)\times V + r \quad (\mathbf{7})$$

To illustrate both theorems 1 and 2, assume a network scenario with n=3 and ns^1=2. Figure 7 shows both Q^1 and R^1 for such scenario. Note that at t_c, the ns^1 requests are placed in the outgoing queue in any arbitrarily order. Whichever the ordering, the busy period corresponds to R^1, and therefore, the worst-case response time for a message request in master 1 is (6): $ns^1\times V=2\times V=2\times 3\times H=6\times H$.

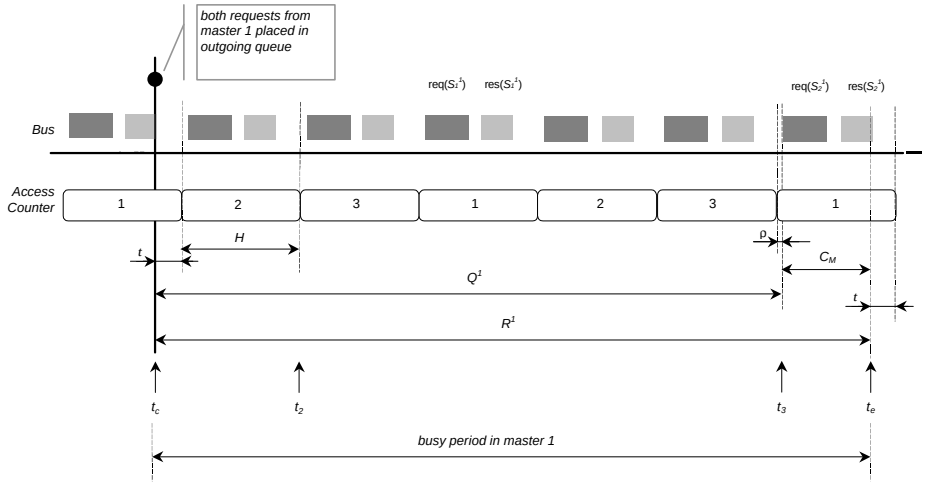

Fig. 7. Queuing and Response Times the Given Scenario

4. Timing Analysis Considering the Actual Token Utilisation

In the previous section we derived a basic timing analysis for the evaluation of the worst-case message response time. Such analysis may however be very pessimistic, since we assumed the token as being fully utilised in the ns^k consecutive token cycles of the busy period. However, the token can only be fully utilised during that interval if:

$$ns^y \geq ns^k, \forall_{y \neq k} \quad (8)$$

as, only in such case, the number of pending requests, in each master *y*, may be greater than ns^k. Otherwise, if $\exists_{y \neq k}$: $ns^y < ns^k$, the token utilisation depends on the periodicity of message streams for those masters *y*.

Definition 3: *Master's Eligible Requests – We define the eligible requests of master y ¹ k, as the maximum number of requests generated in that master that will be pending[10] within the busy period of master k.*

If the number of eligible requests of master *y* (Er^y) is smaller than ns^k, then it is possible that such master will not use all ns^k token visits to process message cycles. Therefore, the evaluation the eligible requests of each master *y*, is paramount for the worst-case response time analysis considering the actual token utilisation. We will use the following equation as the starting basis for the evaluation of Er^y:

[10] Even they are processed during the busy period of master *k*, for a while, they were pending.

$$Er^{y}(t) = ns^{y} + \sum_{i=1}^{ns^{y}} \left\lfloor \frac{t}{T_i^{y}} \right\rfloor \quad (9)$$

Equation (9) gives the maximum number of requests generated by a master y within a time interval t: ns^y requests are made at the beginning of the interval, and then, new requests are made at their maximum rate. This is also known as the *asap* (as soon as possible) pattern (Liu and Layland, 1973). By tailoring equation (9) to encompass the P-NET MAC characteristics, we will be able to perform a worst-case message response time analysis which considers the actual token utilisation, instead of assuming the fully utilisation of the token.

4.1. Concept of P-NET Logical Ring Request Jitter

From equation (9), it is obvious that the larger is considered the time interval, the higher is the value for Er^y. Note, however, that this equation is a step function, hence varying only for multiples of T_i^y.

Consider that in each master $y \neq k$, ns^y requests are simultaneously made at the critical instant (t_c) of master k. Remembering that the busy period is defined as $[t_c, t_e]$, it would be reasonable to consider as eligible requests of master y, all those requests given by $E_r^y = ns^y + \Sigma_{i=1,..,ns^y} \lfloor (t_e - t_c)/T_i^y \rfloor$. In the following analysis, we will show that the worst-case situation appears when the ns^y requests are not simultaneously made in all masters $y \neq k$, and that the quantity before t_c that must be considered for each master y, depends of its logical ring position.

Assume that the critical instant, which must be considered, for each master y, is denoted as t_r^y, with $t_r^y < t_c$, $\forall_{y \neq k}$. Basically we need to analyse how much earlier t_r^y can be made, increasing the number of master y eligible requests $E_r^y = ns^y + \Sigma_{i=1,..,ns^y} \lfloor (t_e - t_c)/T_i^y \rfloor$, without any of the initial ns^y requests being able to be processed in an earlier token visit prior to the critical instant in master k.

For master k^{-1}, which we denote as the predecessor of master k, t_r can be shifted back by $C_M + r + t$, being coincident to the starting of a busy period in master k^{-1}. t_r can not be shifted further back, since it would imply a deadline violation in master k^{-1} or one of the initial ns^y requests would be processed prior to the busy period in master k. Considering t as the token passing time implies that a message cycle was just completed at instant t_r. Otherwise, the token passing time would be reduce to s. Consequently, the total amount t_r may be shifted back for the case of master k^{-1}, is $C_M + r + t = H$.

Considering master k^{-2} and following a similar analysis, t_r could be shifted back up to $2 \times H$. Thus, a different value for t_r, denoted as t_r^y, must be considered for each master $y \neq k$, and its value only depends on the relative logical ring position of master y in respect to master k.

Definition 4: *Logical Ring Request Jitter – We define the logical ring request jitter of master y, as the difference* $Jr^y = t_r^y - t_c$, *being* t_r^y *how much earlier than the critical instant in k, a master y can made its* ns^y

requests, without violating a deadline, nor processing any of those ns^y requests prior to the critical instant in master k.

Resulting from the previous analysis, Jr^y can be expressed as follows:

$$Jr^y = \sum_{i=y,y^{+1},\ldots,k^{-1}} H \tag{10}$$

which, is equivalent to:

$$Jr^y = \left[\left(n + k - y\right) \bmod n\right] \times H \tag{11}$$

To illustrate this definition, assume a network scenario as shown in table 2. For simplification, the periodicity of streams is expressed in multiples of H.

Table 2. Stream Set Scenario 1

Master		**(*C, T, D*)**		
1	$ns^1 = 3$	(C_M, 14, 14)	(C_M, 20, 20)	(C_M, 20, 20)
2	$ns^2 = 1$	(C_M, 8, 8)		
3	$ns^3 = 3$	(C_M, 14, 14)	(C_M, 20, 20)	(C_M, 20, 20)
4	$ns^4 = 3$	(C_M, 14, 14)	(C_M, 20, 20)	(C_M, 20, 20)

The Gantt chart for the master 1 busy period will result as illustrated in figure 8, where t_r^2–t_c, t_r^3–t_c and t_r^4–t_c represent the logical ring request jitter, respectively of masters 2, 3 and 4. Note that for master 2, its number of eligible requests is greater than ns^2.

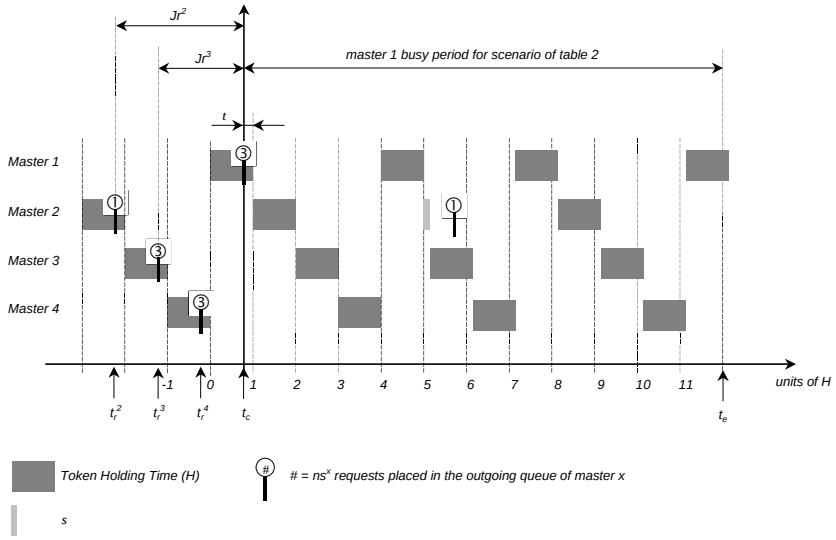

Fig. 8. Busy Period of Master 1 with the Scenario of Table 2

Theorem 3: *In P-NET networks, the longest busy period of master k occurs when all k predecessors started their busy periods in the token cycle previous to the busy period in master k.*

Proof:

The worst-case length busy period in master *k*, results if all the eligible requests of each master *y* are considered for transmission during the interval $t_e - t_c$. Considering that in each master *y*, ns^y requests are placed in each one's outgoing queue at the instant $t_c - Jr^y$, then for each master *y* the number of eligible requests is $E_r^y = ns^y + \Sigma_{i=1,..,ns^y} \lfloor (t_e - t_c + t_r^y)/T_i^y \rfloor$.

Using the definitions of busy period and of logical ring request jitter, if ns^y requests are placed in the outgoing queues at $t_1 - Jr^y$, then, in the token cycle prior to the busy period in master *k*, busy periods has started in all predecessors of *k*.

❑

Considering P-NET's logical ring request jitter concept, the number of eligible requests (9) can now be updated to:

$$Er^y = ns^y + \sum_{i=1}^{ns^y} \left\lfloor \frac{t_e - t_c + t_r^y}{T_i^y} \right\rfloor \qquad (12)$$

which, using theorems 1 and 3 and definition 4, can be re-written as:

$$Er^y = ns^y + \sum_{i=1}^{ns^y} \left\lfloor \frac{R^k + Jr^y}{T_i^y} \right\rfloor \qquad (13)$$

with Jr^y as defined in (11).

4.2. Concept of P-NET Logical Ring Visit Jitter

Equation (13) is quite pessimistic, since not all the eligible requests will be able to be processed within the busy period of master *k*. The reason is obvious. If ns^y is greater than ns^k, only a maximum of ns^k requests will be processed by master *y* within the busy period. However, for our analysis, the relevant case is when $ns^y < ns^k$, since it leads to a scenario where the token is not fully utilised. For this case, even if Er^y (as given by (13)) is larger than ns^k, it might happens that a number smaller than ns^k requests are able to be processed during the busy period.

Intuitively we can show that if a new request appears in the outgoing queue of master *y* and an instant t_2 ($t_c < t_2 < t_e$), this request may not be processed before t_e, even if the outgoing queue in *y* was empty. This is the case of all the requests made in master *y* after the last token visit (to *y*) prior to the completion of the busy period in master *k*.

Assume the following example, where we update in table 2 the periodicity of stream S_1^2 from 8 to 12. The Gantt chart for the busy period in master *k* would be as shown in figure 9, instead of that shown in figure 8. Note that a new request for master 2 appearing before t_e, can not be processed during the busy period of master 1.

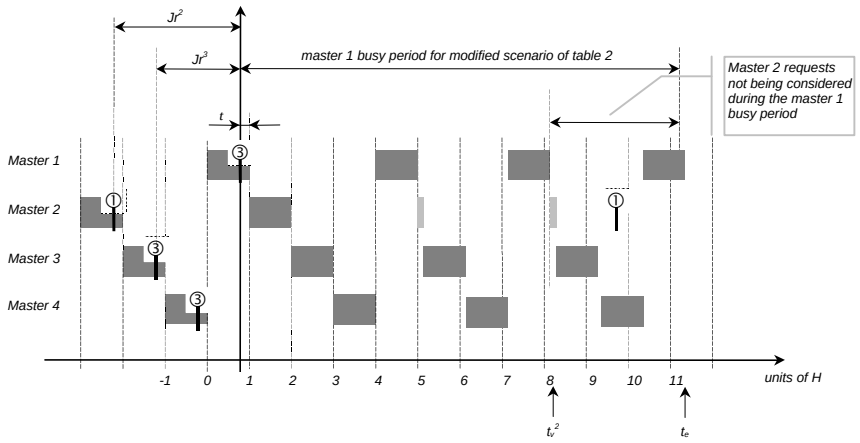

Fig. 9. Busy Period for Master 1 with the Modified Scenario of Table 2

Definition 5: *Processing Window of Master's Busy Period – We define the processing window of master k busy period, as the time span between* t_c *and* t_v^y ($t_v^y < t_e$)*, within which, a first-positioned pending request in master y will assuredly be processed.*

Definition 6: *Logical Ring Visit Jitter – We define the logical ring visit jitter* (Jv^y) *of master y, as the difference* $t_e - t_v^y$.

It becomes obvious that the worst-case response time of a message request in a master k corresponds to processing windows in masters $y \neq k$ as large as possible, since this is the case where more eligible requests would be processed during the busy period of master k. Thus, the worst-case response time of a message request in a master k corresponds to the minimum logical ring visit jitter in masters y.

Such minimum logical ring visit jitter Jv^y can be evaluated considering that none of the masters y processed any message request in the last token visit prior to the completion of the busy period in master k.

Therefore,

$$Jv^y = [((n+k-y) \bmod n) - 1] \times s + r + C_M + (s - r) \quad \textbf{(14)}$$

where $r+C_M$ corresponds to the processing time of the last of the ns^k requests in master k (see figure 3), $s+r$ corresponds to master y, and $[((n+k-y) \bmod n) - 1]$ corresponds to the number of masters between y and k.

There is a certain level of pessimism in considering that none of the masters y processed any message request in the last token visit prior to the completion of the busy period in master k. In fact, if for some of those masters $ns^y \geq ns^k$, then, they will assuredly use the token in all ns^k consecutive cycles of the busy period in master k. For those masters, we may consider H instead of s, hence diminishing the length of the busy period processing window.

Therefore, equation (14) can be updated to:

$$Jv^y = [((n+k-y) \bmod n)] \times s + C_M + \sum_{\substack{i=y^{+1},\ldots,k^{-1} \\ with \\ ns^i \geq ns^k}} (H - s) \qquad (\mathbf{15})$$

For the previous example (figure 9), the number of eligible requests of master 2 would be 2. However, as, for that master, the logical ring visit jitter is $Jv^2=[((4+1-2) \bmod 4)-1]\times s+C_M+\Sigma_{i=3,4 \text{ with } ns^i \geq ns^1}(H-s)=3\times s+C_M+2\times(H-s)=2\times H+ C_M+s$, only one of those two eligible requests is able to be processed within the busy period of master 1.

4.3. Number of Unused Tokens During the Longest Busy Period

During the previous analysis, we are now able to evaluate the maximum number of eligible requests from each master *y* that may be processed during the busy period of master *k*. Such maximum number will lead to the worst-case response time of a message request in master *k*.

Definition 7: *Master's Logical Ring Aggregate Jitter - We denote $Ja^y=Jr^y-Jv^y$ as the logical ring aggregate jitter of master y.*

Definition 8: *Minimum Number of Unused Tokens During a Busy Period - We define the minimum number of unused tokens by a master y (Ut^y) during the busy period in master k, as the minimum number of times that a master y receives the token and does not have any pending requests, during that period.*

Theorem 4: *The minimum number of unused tokens by a master y within a busy period of master k, is $Ut^y=ns^y-min\{ns^k, ns^y+S_{i=1,..,ns^y} \ddot{e}(R^k+Ja^y)/T_i^y \hat{u}\}$.*

Proof:

By theorem 3, the maximum number of eligible requests of master *y* is $ns^y+ +\Sigma_{i=1,..,ns^y}\lfloor(R^k+Jr^y)/T_i^y\rfloor$. From these requests, only those which arrive within the master *k* processing window, will be able to be processed within the busy period. Therefore, the evaluation interval for the *asap* pattern is $R^k+Jr^y- Jv^y= R^k+Ja^y$.

In a master *k*, the number of token cycles during the busy period is, by definition 1, ns^k. Thus, the actual token utilisation by a master *y*, during the busy period of master *k*, is $\min\{ns^k, ns^y+\Sigma_{i=1,..,ns^y}\lfloor(R^k+Ja^y)/T_i^y\rfloor\}$.

As a consequence, the number of times master *y* does not use the token during the busy period of master *k* is:

$$Ut^y = ns^k - \min\left\{ns^k, ns^y + \sum_{i=1}^{ns^y}\left\lfloor \frac{\left(R^k + Ja^y\right)}{T_i^y} \right\rfloor\right\} \qquad (\mathbf{16})$$

❑

Theorem 5: *The minimum number of unused tokens during a busy period of master k, is*

$$Ut = \sum_{y=1}^{n} Ut^{y} \tag{17}$$

Proof:

As for $y = k$, $Ut^{y} = 0$, the proof for this theorem is obvious.

❑

4.4. Analysis of the Worst-Case Response Time

Considering that the token is fully utilised (section 3), the worst-case response time of a message request in a master k (equation (6)) is $R^k = ns^k \times V$. It is now possible to update (6) to incorporate the actual token utilisation, considering that, for each unused token we must subtract the corresponding value of the token holding time (H), and add a s corresponding to the token passing time for the case of an unused token:

$$R^k = ns^k \times V - Ut \times (H - s) \tag{18}$$

Using the results obtained in the previous sub-sections, the worst-case response time of a message request in a master k is:

$$R^k = ns^k \times V - \left[\sum_{y=1}^{n} \left(ns^k - \min \left\{ ns^k, ns^y + \sum_{i=1}^{ns^y} \left\lceil \frac{R^k + Ja^y}{T_i^y} \right\rceil \right\} \right) \right] \times (H - s) \tag{19}$$

As expected, this equation embodies a mutual dependence, since R^k appears in both sides of the equation. In fact, all the previous analysis underlay this mutual dependence, since in order to evaluate R^k, Ut must be found, and *vice-versa*.

The easiest way to solve equation (19) is to form a recurrence relationship (Audsley *et al.* 1993):

$$W^{m+1} = ns^k \times V - \left[\sum_{y=1}^{n} \left(ns^k - \min \left\{ ns^k, ns^y + \sum_{i=1}^{ns^y} \left\lceil \frac{W^m + Ja^y}{T_i^y} \right\rceil \right\} \right) \right] \times (H - s) \tag{20}$$

The set of values $\{W^0, W^1, W^2, \ldots, W^m, \ldots\}$ is monotonically non decreasing, since as W evolves, less unused tokens are being considered. Starting with $W^0 = 0$, when $W^m = W^{m+1}$, the solution of equation (19) has been found.

4.5. Pre-Run-Time Schedulability Condition

Having found the value for the worst-case response time of a message request in each master k, a pre-run-time schedulability test results:

$$D_i^k \geq R^k, \forall_{i,k} \tag{21}$$

4.6. Numerical Example

Assume the stream set shown in table 3.

Table 3. Another Stream Set Example

Master		**(*C, T, D*)**		
1	$ns^1 = 3$	(C_M, 14, 14)	(C_M, 20, 20)	(C_M, 40, 40)
2	$ns^2 = 1$	(C_M, 12, 12)		
3	$ns^3 = 3$	(C_M, 14, 14)	(C_M, 20, 20)	(C_M, 20, 20)
4	$ns^4 = 2$	(C_M, 14, 14)	(C_M, 20, 20)	

Applying equation (19) by using the recurrence relationship given by equation (20), we will be able to find the worst-case response time for master 1.

As the number of streams in master 3 is equal to the number of streams in master 1, we need only to focus on the unused tokens of masters 2 and 4.

Therefore, the network aggregate release jitter for master 2 will be:

$$Ja^2 = 3\times H - Jv^2 = 3\times H - (3\times s + C_M + 1\times(H - s)) = 2\times H - C_M - 2\times s$$

and the network aggregate release jitter for master 4 will be:

$$Ja^4 = 1\times H - Jv^2 = H - (s + C_M + 0) = H - C_M = r + t$$

For $W^0 = 0$, then, W^1 is,

$$= 3\times 4\times H - nut\times(H - s) = 3\times 4\times H - (2+1)\times(H - s) = 9\times H + 3\times s$$

For $W^1 = 9\times H + s$, W^2 is:

$$= 3\times 4\times H - (2+1)\times(H - s) = 9\times H + 3\times s$$

The iterations stop here, as $W^2 = W^1$. This corresponds to Gantt chart illustrated in figure 10.

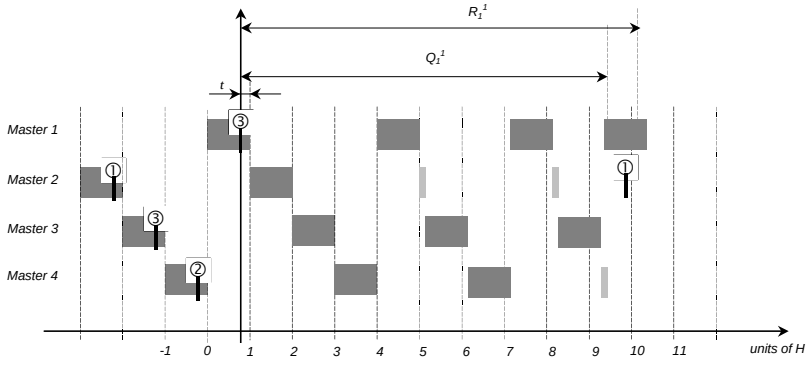

Fig. 10. Busy Period of Master 1 within the Scenario of Table 3

If we consider that the longest message cycle is composed by 67 P-NET frame bytes (request + response), then this corresponds to 67×11=670 bits. Including 30 *bp*, for the worst-case reaction time of a slave, then C_M=767/76800=10 ms.

Therefore, H=(7+767+40)/76800 = 10.6 ms.

This means that the worst-case response time for a message request in master 1 for the scenario of table 3 is: 9×10.6+3×(10/76800)=95.79 ms.

4.7. Considering the Actual Transmission Times for Message Cycles

In the previous analysis, the message cycles' length was considered, for simplification, to be constant at C_M. The results can now be updated, considering the actual message cycles' length, or at least, the longest (smallest) message cycle in each master.

Considering $M^k=\max_{i=1,..,ns^k}\{C_i^k\}$ as the longest message cycle in a master k, then, the worst-case response time of a message request in master k (updating (6)), considering that the token is fully utilised, is (note that now $R_i^k = R^k$, $\forall_i$ is not valid any more):

$$Q_i^k = ns^k \times \sum_{l=1}^{n} \left(r + M^l + t \right) - C_i^k \tag{22}$$

The logical ring request jitter (10), can be updated to:

$$Jr^y = \sum_{i=y,y^{+1},...,k^{-1}} \left(r + M^i + t \right) \tag{23}$$

The logical ring visit jitter (15), can be updated to:

$$Jv^y = \sum_{l=y,..,k^{-1}} s + \min_{i=1,..,ns^k} \left\{ C_i^k \right\} + \sum_{\substack{l=y^+,..,k^{-1} \\ ns^l \geq ns^k}} \left(L^l - s \right) \tag{24}$$

where L^l is defined as the smallest message cycle in a master l: $L^l=r+\min_{i=1,...,ns^l}\{C_i^l\}+t$.

The worst-case response time of a message request in a master k (19) can be updated to (we need now to consider the shortest holding time in each master):

$$R_i^k = ns^k \times V - \sum_{y=1}^{n} \left[\left(ns^k - \min \left\{ ns^k, ns^y + \sum_{j=1}^{ns^y} \left\lfloor \frac{R_i^k + Ja^y}{T_j^y} \right\rfloor \right\} \right) \times \left(L^y - s \right) \right] \tag{25}$$

Finally, as R_i^k may be different from stream to stream in each master, the pre-run-time schedulability condition can be updated to:

$$D_i^k \geq R_i^k, \forall_{i,k} \tag{26}$$

4.8. Pre-Run-Time Schedulability Tool

As for the case of the response time analysis in a single processor environment, the communication response time analysis has the same drawback, which is that each message stream must be individually tested.

However, since the schedulability condition proposed in this paper is to be done prior to run time, no major obstacle exists, provided that a software analysis tool is available.

In Annex A, we outline an algorithm, which is the basis for the implementation of such software tool. The algorithm presented considers always C_M instead of the actual

length for message cycles. Upgrade to include results from equation (22) to (26) is a straightforward procedure.

5. Conclusions

In this paper we have drawn a comprehensive study on how to use P-NET to support real-time communications in distributed computer-controlled systems. The major contribution is to provide a less pessimistic evaluation of the worst-case response time in P-NET networks, as we consider the actual token utilisation during the busy period of masters, whereas previous work considered a fully utilisation of the token during that period. We introduced the concept of P-NET Logical Ring Aggregate Jitter, which is paramount for the evaluation of the actual token utilisation during a master's busy period.

With this analysis we are able to guarantee more stringent relative deadlines of P-NET Messages. Indirectly, we can also say that with the analysis proposed in this paper we can guarantee message streams with shorter periods, hence allowing a higher utilisation of the real-time communication network. This is important, as it is known that in order to guarantee schedulability for the systems' peak load, the average utilisation of the system becomes very poor.

References

Agrawal, G., Chen, B., Zhao, W. and Davari, S. 1992. Guaranteeing Synchronous Message Deadlines with the Timed Token Protocol. Proceedings of the 12th IEEE International Conference on Distributed Computing Systems.

Audsley, N., Burns, A., Richardson, M., Tindell, K. and Wellings, J. 1993. Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. Software Engineering Journal, 8(5): 284-292.

Cenelec 1996. General Purpose Field Communication System, Vol. 1/3 (P-NET), Vol. 2/3 (Profibus), Vol. 3/3 (FIP). CENELEC.

Fip 1990. Normes FIP NF C46-601 to NF C46-607. Union Technique de l'Electricité, AFNOR.

Grow, R. 1982. A Timed Token Protocol for Local Area Networks. Proceedings of Electro'82, Token Access Protocols, Paper 17/3.

Jenkins, C. 1997. P-NET as a European Fieldbus Standard EN 50170 vol. 1. The Institute of Measurement + Control Journal, 1997.

Liu, C. and Layland, J. 1973. Scheduling Algorithms for Multiprogramming in Hard-Real-Time Environment. Journal of the ACM, 20(1): 46-61.

Montuschi, P., Ciminiera, L. and Valenzano, A. 1992. Time Characteristics of IEE802.4 Token Bus Protocol. IEE Proceedings, 139 (1): 81-87.

Pedro, P. and Burns, A. 1997. Worst Case Response Time Analysis of Hard Real-Time Sporadic Traffic in FIP Networks. Proceedings of 9th Euromicro Workshop on Real-time Systems. Toledo, Spain, pp. 5-12.

P-net 1994. The P-NET Standard. International P-NET User Organisation ApS.

Profibus 1992. Profibus Standard DIN 19245 part I and II. Translated from German, Profibus Nutzerorganisation e.V.

Raja, P., Ruiz, L. and Decotignie, J.-D. 1995. On the Necessary Real-Time Conditions for the Producer-Distributer-Consumer Model. Proceedings of 1st IEEE Workshop on Factory Communication Systems. Leysin, Switzerland.

Sae 1992. SAE J1583, Controller Area Network (CAN), an In-Vehicle Serial Communication Protocol. SAE Handbook, Vol. II.

Stankovic, J. 1988. Real-Time Computing Systems: the Next Generation. Stankovic J., Ramamritham, K. (Eds.), IEEE Press. Tutorial: Hard Real-Time Systems: pp. 14-38.

Tindell, K., Hansson, H. and Wellings, A. 1994. Analysing Real-Time Communications: Controller Area Network (CAN). Proceedings of the IEEE Real-Time Systems Symposium. San Juan, Puerto Rico, pp. 259-263.

Tindell, K., Burns, A. and Wellings, A. 1995. Analysis of Hard Real-Time Communications. Journal of Real-Time Systems, 9, Kluwer.

Tovar, E. and Vasques, F. 1998a. Guaranteeing Real-Time Message Deadlines in Profibus Networks. Proceedings of the 10th Euromicro Workshop on Real-time Systems. Berlin, Germany, pp. 79-86.

Tovar, E. and Vasques, F. 1998b. Setting Target Rotation Time in Profibus Based Real-Time Distributed Applications. Proceedings of the 15th IFAC Workshop on Distributed Computer Control Systems. Como, Italy, pp. 1-6.

Tovar, E., Vasques, F. and Burns, A. 1998c. Real-Time Communications in Multihop P-NET Networks. Submitted to Control Engineering Practice. Pergamon.

Annex A

```
Algorithm pnet_sched_analysis
input: n          /* number of masters */
       pass       /* time to pass the token after message cycle */
       idle       /* time to pass the token if no message cycle transmitted */
       ns[w]      /* array containing number of streams in each master; w ranges from 1 to n */
       M[x, y, z] /* message streams information; x ranges from 1 to n;y ranges from 1 to max (ns[])*/
                  /* z ranges 1 to 3; z = 1(len. of mes. cycle); z = 2 (period); z = 3 (relative deadline) */
output:O[x, y]    /* similar to M [x, y, z] except for z */
                  /* if O[x, y] = 1 stream marked as not schedulable; if O[x, y] = 0 stream schedul. */
       R[x, y]    /* worst-case response time; x ranges from 1 to n;  y ranges from 1 to max (ns[])*/
begin
 1:    /* computation of CM */
 2:    CM = 0;
 3:    for i = 1 to n do
 4:        for j = 1 to ns[i] do
 5:            if M[i, j, 1] > CM then
 6:                CM = M[i, j, 1]
 7:            end if
 8:        end for
 9:    end for
 10:   H = react + CM + pass;
 11:   for i = 1 to n do
 12:       R_tdma = ns[i] * n * H;
 13:       R = 0;
 14:       repeat
 15:           R_Before = R; unt = 0;
 16:           for j = 1 to n do
 17:               if j <> i then
 18:                   /* computation of visit jitter */
 19:                   jv = calc_visit (i, j)
 20:                   /* computation of aggregate jitter */
 21:                   jitter = ((n + i – j) mod n) * H – jv;
 22:                   add_req = 0;
 23:
 24:                   for l = 1 to ns[j] do
 25:                       add_req = add_req + int ((R + jitter) / M[j, l, 2])
 26:                   end for
 27:                   if (add_req + ns[j]) < ns[i] then
 28:                       unt = unt + (ns[i] – add_req – ns[j])
 29:                   end if
 30:               end if
 31:               R = R_tdma - unt * (H – idle)
 32:           end for
 33:       until R = R_Before;
 34:       for j = 1 to ns[i] do
 35:           R[i, j] = R;
 36:           if M[i, j, 3] < R then
 37:               /* mark message stream j of master i not schedulable */
 38:               O[i, j] = 1
 39:           end if
 40:       end for
 41:   end for
end.
```

```
Function visit_jitter (i, j)
input: i                    /* equivalent master k */
       j                    /* master y */
       H, ns[w], idle       /* global vars */
output: vj                  /* visit jitter */

Begin
 1:     jv = ((n + i – j) mod n) * idle + CM;
 2:     if j > i then
 3:         for k = j + 1 to n do
 4:             if ns[k] >= ns[i] then
 5:                 jv = jv + H – idle
 6:             end if
 7:         end for
 8:         for k = 1 to i –1 do
 9:             if ns[k] >= ns[i] then
10:                 jv = jv + H – idle
11:             end if
12:         end for
13:     else
14:         for k = j + 1 to i –1 do
15:             if ns[k] >= ns[i] then
16:                 jv = jv + H – idle
17:             end if
18:         end for
19:     end if
return jv
```